Softwarenaut

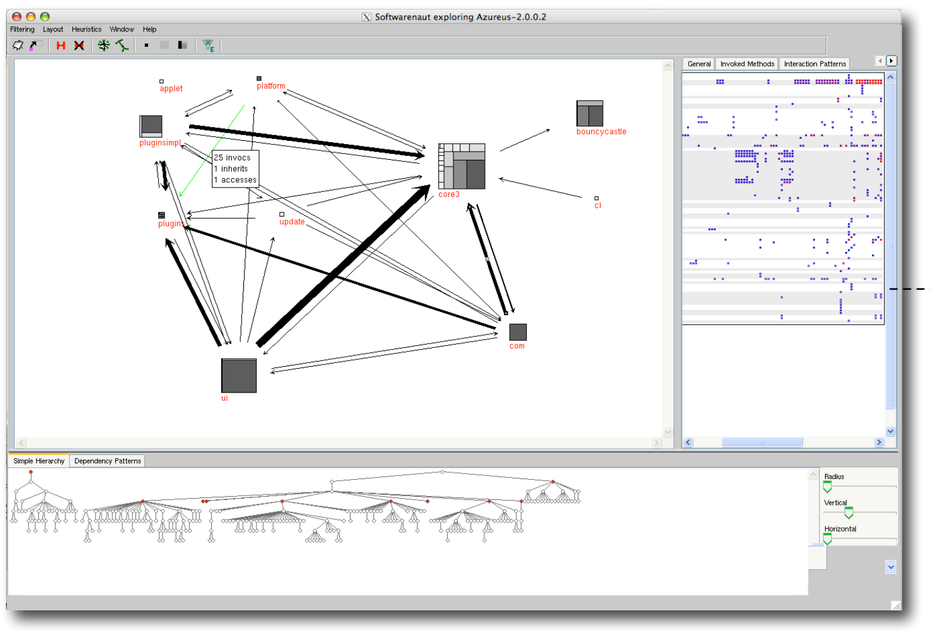
Lungu, Mircea; Lanza, Michele

Published in:
Proceedings of Softvis 2006 (3rd International ACM Symposium on Software Visualization)

Softwarenaut: *Cutting* Edge Visualization in Software

Mircea Lungu and Michele Lanza
Faculty of Informatics, University of Lugano, Switzerland

Softwarenaut is a tool aimed at supporting the interactive exploration of software systems [1]



Provider Module

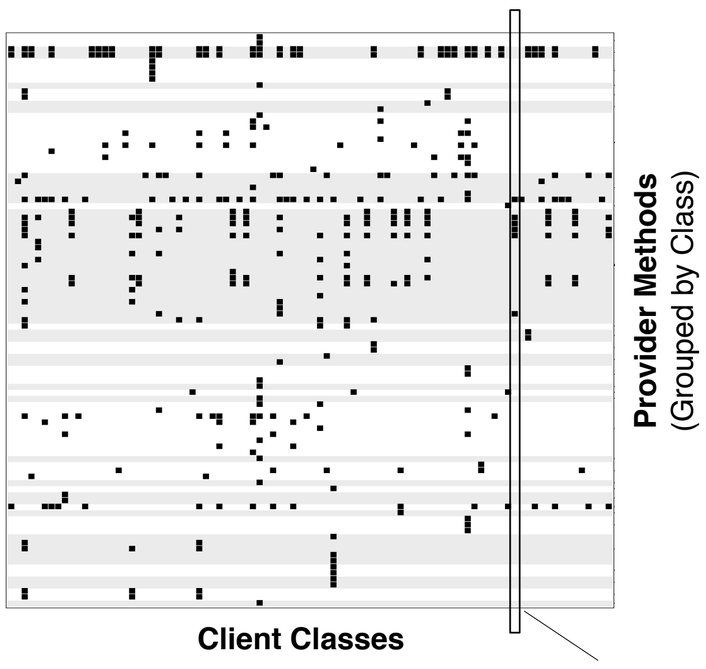
core3

Client Module

ui

How can we devise a visual representation to help the user see what hides *behind* the edge?

One possible representation for a dependency edge is an Incidence Matrix



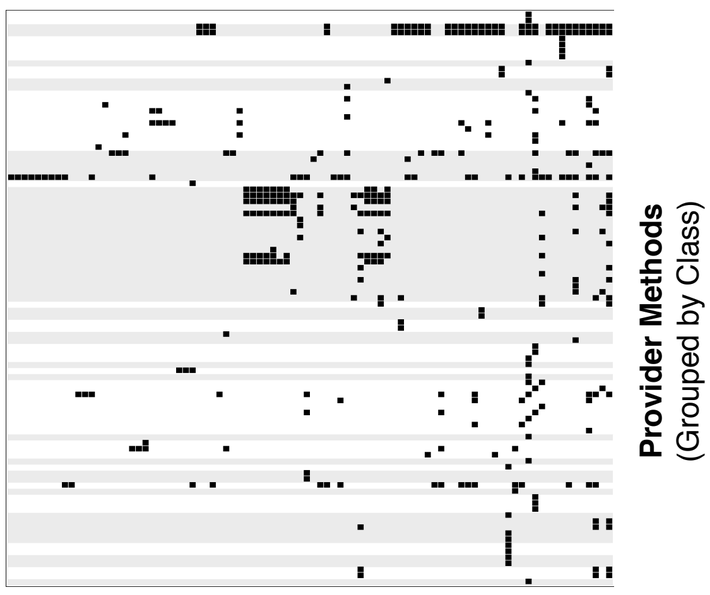
Based on a hierarchical clustering algorithm we reorder the columns in order to group similar ones

A **dependency incidence matrix** is a binary matrix that has client classes as columns and provider methods as rows and has a 1 at the intersection of a method and a class if the class invokes that method at least once. For each class a **dependency signature** is a column in the matrix.

Based on the distances between the signatures, the classes are clustered using a hierarchical clustering algorithm and then the columns of the matrix are reordered. The new order is the result of traversing the resulting dendrogram in preorder. This determines **classes with similar invocation patterns to be disposed in spatial proximity** on the X-axis.

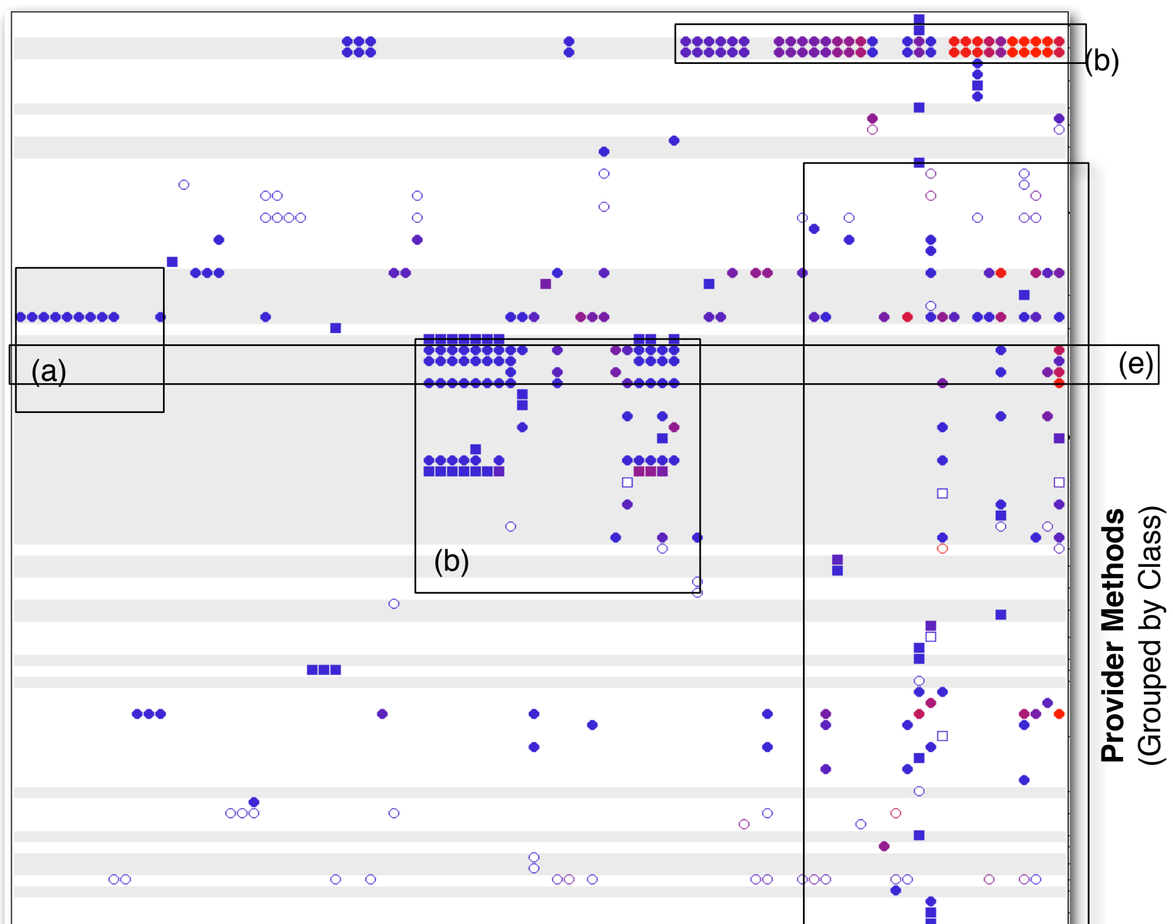
As a side effect of the X-axis ordering, **the classes with stronger coupling are positioned towards the right** side of the matrix. It is among these classes that we usually find the most important classes in the client module.

After reordering the classes, we can visually check for interesting dependency patterns in the data



Enriching the view with structural and semantic information

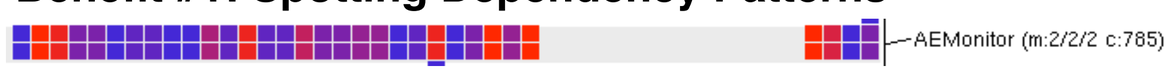
Using **color and shape** we can provide more information about the elements in the view. For example, we can add information about the number of invocations to a given method as well as the method type (see the legend below):



Legend

- blue - low number of invocations from the class to the method
- red - high number of invocations from the class to the method
- empty geometrical shape - accessor method (get*,set*)
- **filled geometrical shape** - other methods

Benefit #1: Spotting Dependency Patterns

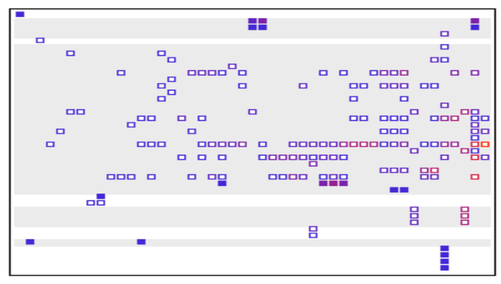


azureus: com->org.gudy.azureus.core3

Many classes interact with the two methods above in the very same manner. Each time one is called the other is also called. It turns out that they are a pair of methods that have to be always called together. If we would see an incomplete pattern (i.e. a class which does not call both the methods, we could suspect a defect)

Benefit #2: Characterizing (some) dependencies at a glance

Some special types of **dependency patterns are easy to detect visually but hard to detect automatically**. The dependency from right would not be detected as a data dependency (the client requests data from the provider as there are many accessor methods). Because not all the methods are accessors this would be hard to automatically detect.



Benefit #3: Interactivity

In the context of Softwarenaut, the matrix **is a starting point** for further exploration and manipulation of the represented entities (e.g. filtering, detailing). The view is not standalone but is **closely integrated** with the Softwarenaut environment. Requesting **detail about a given entity** or a group of entities **is one click away**. The code of the classes and methods is also reachable from each entitty.

Drawbacks

The data is usually noisy and **patterns are not always easy to detect**. There matrix presents also a **scalability problem**: the screen can not always contain the whole matrix.

References

[1] Mircea Lungu and Michele Lanza. - "Softwarenaut: Exploring hierarchical system decompositions". In Proceedings 10th European Conference on Software Maintenance and Reengineering (CSMR 2006), IEEE Computer Society Press,

For more information see: http://www.inf.unisi.ch/phd/lungu/softwarenaut